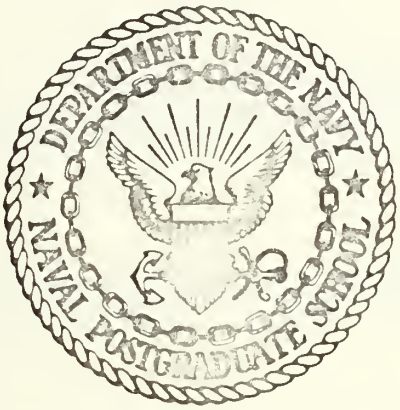
SEQUENTIAL DECODING
OF
TIME-VARYING CONVOLUTIONAL CODES

Barry Harold Lerich

United States
Naval Postgraduate School



THESIS

SEQUENTIAL DECODING
OF
TIME-VARYING CONVOLUTIONAL CODES

by

Barry Harold Lerich

Thesis Advisor: J. M. Geist

December 1971

Sequential Decoding of Time-varying Convolutional Codes

by

Barry Harold Lerich
Lieutenant Commander, United States Navy
B.S., Purdue University, 1964

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1971

ABSTRACT

A discussion of convolutional encoding and sequential decoding is given as background for the experimental phase of this research. A modified stack algorithm is programmed to simulate a sequential decoder for use in the study of time-varying convolutional codes.

The decoder simulator is used to analyze the structure of codes with the aim of finding a systematic method by which to construct time-varying convolutional codes with good decoding properties. Some difficulties in construction techniques are discussed.

Extensive computer simulation comparing decoding performance of time-varying codes to that of fixed codes is reported. The conclusion of these comparisons is that the time-varying codes studied thus far are of theoretical interest only.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENT

I wish to express my sincere gratitude to Professor John M. Geist without whose patience and guidance this work would have been impossible.

In addition, I wish to acknowledge the helpful suggestions received in the technical aspects of the computer simulations from Mr. Robert Limes and Mr. Al Wong of the Electrical Engineering Department Computer Laboratory.

I. INTRODUCTION

A. BACKGROUND

Since noise is always present in electrical communications circuits, there is a non-zero probability that transmitted signals will be altered causing errors at the receiver. When digital information is transmitted directly from machine to machine, erroneous reception of even a single bit of information may change the complete meaning of a message. It is for this reason that error-correcting codes have been devised. These codes should provide a means of communicating with any required degree of reliability, and should be easily implemented with existing digital equipment.

Shannon [1] has shown that codes meeting the first criterion do exist provided that the data rate does not exceed some maximum, called channel capacity. The statistical characteristics of the space channel have indicated that the use of convolutional codes with sequential decoding is one of the best methods of attaining the reliability required over this type of channel. In this chapter a general convolutional encoder and sequential decoding are briefly discussed. A description of the channel model used in simulation studies is also given.

B. CONVOLUTIONAL CODES

Forney [2] has given the following definition: "An (n,k) convolutional encoder over a finite field, F, is a k-input n-output constant linear causal finite-state sequential circuit." The rate of a general convolutional encoder is defined to be k/n.

A general binary rate ½ convolutional encoder of memory m is shown in Figure 1. The input, $\{x_k\}$, is a binary sequence, called the information sequence, and the outputs are two binary sequences, $\{y_k^{(1)}\}$ and $\{y_k^{(2)}\}$, either commutated to form a single sequence or transmitted independently. At each unit of time the encoder transforms one input digit into a sequence of two output digits. It can be seen from Figure 1 that the two output digits depend on the present information digit and the m previous information digits.

The transform A(D) of the sequence $a_0,a_1,a_2,\ldots$ is defined as a formal power series in the indeterminate, D.

$$A(D) = a_0 + a_1D^1 + a_2D^2 + \ldots$$

If the sequence $a_0,a_1,a_2,\ldots$ has only a finite number of non-zero elements then A(D) is a polynomial in D. The D-transforms of the sequences $g_0^{(i)},g_1^{(i)},g_2^{(i)},\ldots,g_m^{(i)}$,

$$G_1(D) = g_0^{(1)} + g_1^{(1)}D + g_2^{(1)}D^2 + \ldots + g_m^{(1)}D^m$$

$$G_2(D) = g_0^{(2)} + g_1^{(2)}D + g_2^{(2)}D^2 + \ldots + g_m^{(2)}D^m,$$

are called the code generating polynomials of the encoder. In Figure 1, if $g_k^{(i)} = 0$, no wire is present and, if

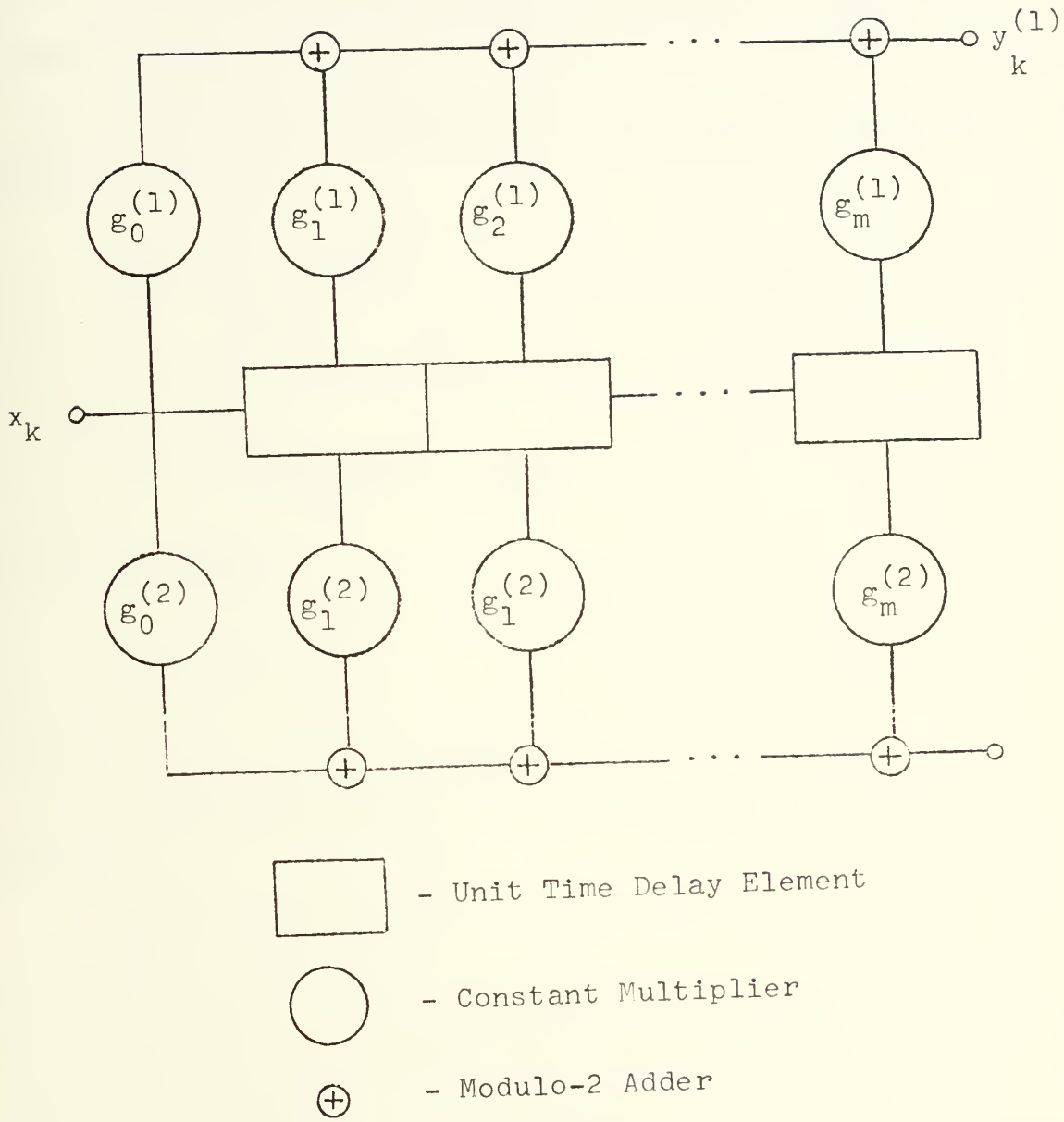


Figure 1. General Binary Rate ½ Convolutional Encoder.

$g_k^{(i)} = 1$, a wire is present. If the generating polynomials are time-invariant, the encoder is known as a fixed convolutional encoder.

One approach to looking at the encoder is noting the outputs at time k.

$$y_k^{(j)} = \sum_{i=0}^{m} g_i^{(j)} x_{k-i} , \qquad j = 1,2 \quad \text{(modulo-2 sum)}$$

This is the convolution of the sequence

$$g^{(j)} = g_0^{(j)}, g_1^{(j)}, \ldots, g_m^{(j)}$$

with the sequence

$$x_o, x_1, x_2, \ldots ;$$

hence the name "convolutional" code.

The D-transform, $Y_j(d)$, of the sequence $\{y_k^{(j)}\}$ is given by

$$Y_j(D) = G_j(D)X(D) , \qquad j = 1,2$$

where X(D) is the transform of the input sequence. Therefore, as in continuous linear systems, convolution in the time domain corresponds to multiplication in the D-transform domain, where, for binary systems, the multiplication is performed modulo-2.

Another convenient way to view convolutional encoders is through a matrix description. Using Forney's stipulation [2] that a convolutional encoder is a causal linear circuit, it can be assumed that the information sequences begin at

time zero. Therefore, for a rate ½ fixed convolutional encoder, the information sequence is

$$\underline{X} = \left[x_0, x_1, x_2, \ldots\right] ,$$

and the corresponding output sequence is

$$\underline{Y} = \left[y_o^{(1)} y_o^{(2)}, y_1^{(1)} y_1^{(2)}, y_2^{(1)} y_2^{(2)}, \ldots\right] .$$

The encoding equations for the encoder can be written

$$\underline{Y} = \underline{X}\ \underline{G},$$

where for an encoder with memory m,

$$\underline{G} = \begin{bmatrix} \underline{G}_o & \underline{G}_1 & \underline{G}_2 & \cdots & \underline{G}_m & \underline{0} & \underline{0} & \cdots \\ \underline{0} & \underline{G}_0 & \underline{G}_1 & \cdots & \underline{G}_{m-1} & \underline{G}_m & \underline{0} & \cdots \\ \underline{0} & \underline{0} & \underline{G}_0 & \cdots & \underline{G}_{m-2} & \underline{G}_{m-1} & \underline{G}_m & \cdots \\ & & & \cdot & & & & \\ & & & \cdot & & & & \\ & & & \cdot & & & & \end{bmatrix}$$

is called the generator matrix and

$$\underline{G}_i = \left[g_i^{(1)}\ g_i^{(2)}\right], \qquad 0 \leq i \leq \infty.$$

Encoders utilizing generator polynomials which vary with time, u, are known as time-varying convolutional encoders. If $\underline{G}_i(u) = \underline{G}_i(u + T)$, $0 \leq u < \infty$, the encoder is a periodic time-varying convolutional encoder with period T. The generator matrix for a time-varying, rate ½ encoder with memory m is given by

$$\underline{G} = \begin{bmatrix} \underline{G}_0(0) & \underline{G}_1(1) & \underline{G}_2(2) & \cdots & \underline{G}_m(m) & \underline{0} & \underline{0} & \cdots \\ \underline{0} & \underline{G}_0(1) & \underline{G}_1(2) & \cdots & \underline{G}_{m-1}(m) & \underline{G}_m(m+1) & \underline{0} & \cdots \\ \underline{0} & \underline{0} & \underline{G}_0(2) & \cdots & \underline{G}_{m-2}(m) & \underline{G}_{m-1}(m+1) & \underline{G}_m(m+2) & \cdots \\ & & & \ddots & & & & \end{bmatrix}$$

where $\underline{G}_i(u) = \left[g_i^{(1)}(u) \; g_i^{(2)}(u) \right]$.

Costello [3], p. 81, has given a theorem which indicates that there exists at least one non-systematic, periodic, time-varying code with decoding properties which may be vastly better than those of fixed codes. The theorem, which will be further discussed in Chapter II, is the major factor which prompted the search for such a code in this research project.

C. TREE STRUCTURE OF CONVOLUTIONAL CODES

Now consider the rate ½, memory-2 encoder in Figure 2. The delay units can be a two-stage digital register. In this case

$$G_1(D) = 1$$
$$G_2(D) = 1 + D + D^2$$

and the code generated is called <u>systematic</u>. This term is used for codes which have the property that the information sequence appears explicitly in the output. Had $G_1(D)$ contained other non-zero coefficients of powers of D, the code would have been <u>non-systematic</u>.

To see how the convolutional encoder constructs the coded output codeword, initially set all stages of the

register to zero. Sequentially shift information bits into the register and after each shift, observe the outputs y_1 and y_2. For example, the information sequence 1 0 1 1 ... will yield the encoded sequence 11 01 10 10 ... as the transmitted symbols.

From this example it can be seen that the transmitted symbols depend on the present and two past information bits. The structure of the set of all output codewords can be seen from a code tree, constructed as follows. For an information sequence, $\underline{X} = [x_0, x_1, \ldots, x_{L-1}]$, the set of all 2^L L-component input vectors is diagrammed on an input tree using the convention that the upper branch diverging from a node corresponds to shifting a zero into the encoder register and the lower branch to shifting in a one. The code tree is completed by adding to each branch of the input tree the output digits associated with it. Using the example given previously, the tree of Figure 3 is constructed. It can be seen that each possible information sequence defines a unique path through the tree. The encoded sequence of the example is shown by the bold path in Figure 3. The concept of the tree structure of convolutional codes leads directly to a discussion of sequential decoding.

D. SEQUENTIAL DECODING

Sequential decoding, introduced by Wozencraft[4], is probabilistic in nature. The decoder has available, or is able to calculate, the channel error probability characteristics and a copy of the code tree used by the encoder.

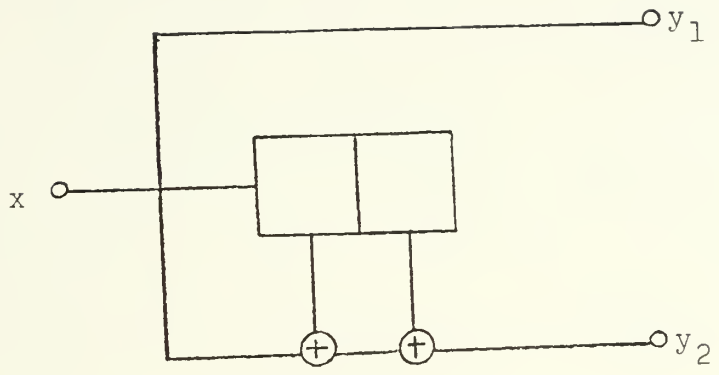


Figure 2. Binary Rate ½ Encoder of Memory Two.

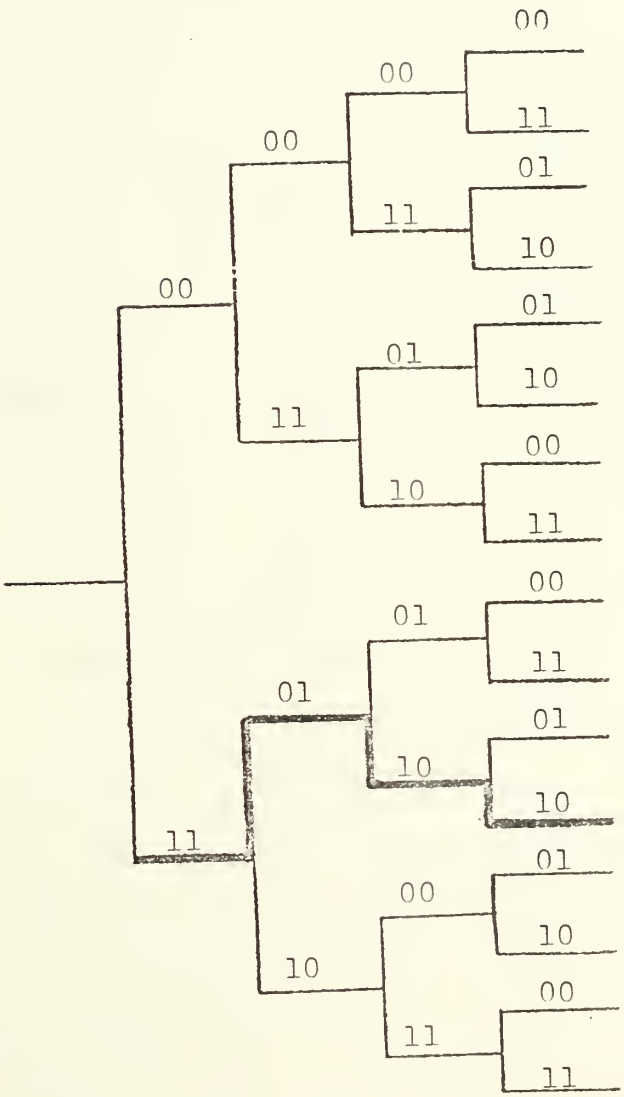


Figure 3. Code Tree for the Encoder of Figure 2.

Sequential decoding algorithms cannot, in practice, construct the complete tree for each received sequence since the number of nodes grows exponentially with the length of the sequence. The decoder in actuality maintains a replica of the convolutional encoder. Wozencraft and Jacobs [5], p. 426, noted that the input, $\underline{X}$, to the encoder may be regarded as a set of instructions which direct the encoder on a unique path through the code tree. Consider a binary symmetric channel (BSC) and a rate ½ encoder. The decoder, with a replica of the encoder, starts at the first node in the code tree, generates both succeeding branches and follows the one that agrees most closely with the received sequence. The process continues until the decoder reaches one of the 2^L terminal nodes for an L-bit information sequence. At this point the sequence is decoded. In the case when the channel is noiseless, the result is trivial, provided that both branches diverging from any one node differ in at least one bit. Such a difference may be guaranteed by ensuring that $g_0^{(i)} = 1$, as can be seen from Figure 2 and the resulting code tree of Figure 3.

When noise is introduced over the channel, the foregoing process is not sufficient to decode a message. In this case, if neither branch emanating from a node corresponds in all places to the respective received bits, the decoder follows whichever branch agrees best. If the decoder makes an error at a particular node, it is unlikely to find a path along subsequent branches which agrees well with the

remainder of the received sequence. At this point the decoder is programmed to go back to the last "best" node and search another path. In practical decoding algorithms, a metric value is assigned to every branch in the tree. If

$$\underline{Y} = \left[y_1, y_2, \ldots, y_n\right]$$

are the code symbols on the branch and

$$\underline{R} = \left[r_1, r_2, \ldots, r_n\right]$$

are the corresponding received symbols, then the branch metric is taken to be

$$a = \sum_{i=1}^{n} \left[\text{Log} \frac{P\{r_i | y_i\}}{p(r_i)} - B \right], \qquad \text{(I.1)}$$

where $P\{r_i|y_i\}$ is the channel transition probability function, $p(r_i)$ is the nominal received symbol probability density function and B is a bias term. Proper choice of B will ensure that, on the average, metric values along the correct path are positive while those along the incorrect path are negative. Nodes are assigned values equal to the sum of the branch metrics along the path leading to the node. The decoder need only look for a path with generally increasing value to find the correct path.

C, the number of computations performed by a sequential decoder, can be defined as the number of nodes examined in the search for a path through the tree. C is a random variable, since the number of computations performed depends upon the received sequence, which in turn depends

upon the channel noise. Jacobs and Berlekamp [6] have shown that the probability distribution function of C behaves as

$$P\{C \geq N\} \geq KN^{-\rho}$$

for large N, where ρ is a number depending only on the channel and the rate of the code, and K varies slowly with N. The distribution is termed as Paretian. This unfortunate behavior of the distribution of computation seems to be the major drawback to the use of sequential decoders. The implication is that decoding speed is a variable and received data must be stored in a buffer as received until it can be processed by the decoder. The Pareto distribution of computations indicates that there will be times when the decoder will fall so far behind that the buffer will overflow. In these cases there are two options:

(1) Divide the information sequence into "frames." When the buffer overflows during a frame, put the frame aside to be processed later and let the decoder start the next new frame.

(2) If overflow occurs, request that the transmitter stop, resynchronize, and begin sending again.

It can be verified that $\bar{C} < \infty$, if and only if $\rho > 1$. The code rate such that $\rho = 1$ is called R_{comp}. The significance of this is that for rates $R < R_{comp}$, $\bar{C}$ is bounded, while for $R > R_{comp}$, $\bar{C} = \infty$. Therefore, decoding is considerably more difficult at rates substantially above R_{comp}. Usually, in sequential decoding, R_{comp} is greater than or equal to one-half channel capacity.

One major advantage to sequential decoding schemes, exemplified by Gallager [7], p. 284, is the fact that the

probability of error decays exponentially with encoder memory m; thus reliability can be increased as much as desired with relatively small increase in encoder complexity, and almost no change in decoder complexity. Another advantage is the relative ease of implementation of sequential decoding systems versus other systems with comparable reliability.

These advantages of sequential decoding weigh so heavily that, in spite of the necessity of operating at rates significantly below capacity, it is generally recognized that systems employing convolutional codes and sequential decoding are the most practical means of achieving high-reliability, low power communication over memoryless channels.

E. BINARY SYMMETRIC CHANNEL

Throughout the experimental phase of this research project, the binary symmetric channel was used as the channel model, and it is shown in Figure 4 for completeness. With probability $q = 1 - p$, the output symbol is a replica of the input symbol, and with probability p, it is the complement of the input. Those crossover probabilities, p, with corresponding R_{comp}, channel capacity, and ρ utilized in simulation are listed in Table I. With $R = \frac{1}{2}$, these choices of p represent operation at approximately R_{comp} and ten percent below R_{comp}.

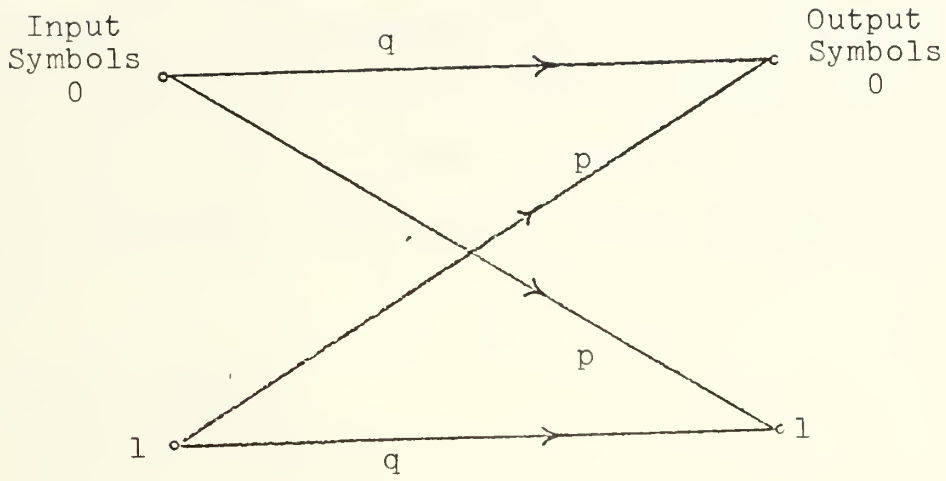


Figure 4. Binary Symmetric Channel.

p	R_{comp}	Capacity	ρ
0.03125	0.562	0.793	1.371
0.04688	0.491	0.727	0.950

Table I. Statistics for Channel Models used in Simulation.

II. DISTANCE DEFINITIONS AND BOUNDS

The distance between codewords in a convolutional code is one measure used in analyzing the error-correcting capabilities of the code. In this chapter, several different distance measures will be defined for convolutional encoders and some established lower bounds on these measures will be shown.

A. DISTANCE MEASURES

1. Minimum Distance

Massey [8], p. 15, has defined the minimum distance, d_{min}, of a convolutional code to be the smallest number of symbols in which two initial codewords differ that do not have identical sets of first information symbols. That is,

$$d_{min} = \min_{\underline{X}_0 \neq \underline{X}'_0} d_H \left([\underline{X}\ \underline{G}]_m, [\underline{X}'\underline{G}]_m \right) ,$$

where $d_H(Z,Z')$ denotes the Hamming distance between the two arguments, $[\underline{Z}]_m$ means the first (m+1) blocks of the sequence $\underline{Z}$, and m is the memory of the encoder. Generalized to time-varying encoders, this becomes

$$d_{min} = \min_{0 \leq u < \infty} \min_{\underline{X}_u \neq \underline{X}'_u} d_H \left([\underline{X}\ \underline{G}]_{u+m}, [\underline{X}'\underline{G}]_{u+m} \right).$$

If the code is periodic with period T, the definition reduces to the following:

$$d_{min} = \min_{0 \leq u < T} \min_{\underline{X}_u \neq \underline{X}'_u} d_H \left([\underline{X}\ \underline{G}]_{u+m}, [\underline{X}'\underline{G}]_{u+m} \right).$$

Minimum distance, so defined, is also known as Feedback Decoding Minimum Distance. The term "feedback" derives from the fact that in some schemes of decoding, error propagation occurs as a result of internal feedback in the decoder. That is, decoded versions of prior sequences are used to decode future sequences and earlier decoding errors can affect future decoding. On the other hand, decoding methods having no internal feedback have been termed definite decoding by Robinson [9].

Costello [3], p. 25, has given an argument to show that, insofar as distance properties are concerned, it is not necessary to consider non-periodic time-varying encoders. Throughout the rest of this chapter, all time-varying encoders will, therefore, be considered to be periodic.

2. Order-j Column Distance

If the encoder input is a binary sequence beginning with a "1", and the encoder output is observed up until time j, the order-j column distance, d_j, is the minimum weight of all such output sequences. For a periodic encoder, d_j is given by

$$d_j = \min_{0 \leq u < T} \min_{\underline{X}_u \neq \underline{X}'_u} d_H \left([\underline{X}\ \underline{G}]_{u+j}, [\underline{X}'\underline{G}]_{u+j} \right)$$

for $j = 0,1,2,\ldots$, and $d_\infty = \lim_{j \to \infty} d_j$. From the foregoing definitions of minimum distance, it can be seen that $d_{min} = d_m$.

3. Free Distance

In a convolutional encoder of memory m, any particular information symbol can affect the output sequences over, at most, (m+1) time units. During this time, for an (n,k) encoder, a total of (m+1)n symbols are transmitted. The code is said to have a feedback decoding constraint length, n_A = (m+1)n symbols. Since sequential decoders are not limited to consider only one constraint length of received digits while attempting to decode a particular block of transmitted digits, a more meaningful distance measure called free distance has become used. The free distance is defined over the entire encoded sequence, and thus is a more appropriate distance measure for a decoder which considers the entire received sequence in making a decision as to which symbols were transmitted.

Free distance has been defined as the minimum weight encoded sequence such that $\underline{X} \neq 0$.

$$d_{free} = \min_{\underline{X} \neq 0} W_H (\underline{X}\ \underline{G})$$

where $W_H(Z)$ denotes the Hamming weight of the argument. It can be shown that $d_{free} = \lim_{j \to \infty} d_j$. Since free distance is a property of the code itself, it appears that this is the distance measure of concern when considering time-varying codes.

B. BOUNDS ON DISTANCE MEASURES

It is desirable from an error-correcting capability standpoint to maximize the particular distance measure

with which one is concerned. Although it is not always clear how to construct codes with good distance properties, it is possible to calculate bounds on these distance measures.

1. Feedback Decoding Minimum Distance for Fixed Codes

Wozencraft and Reiffen [10], p. 51, and Massey [8], p. 15, have shown that there exists at least one fixed binary convolutional code with a given rate R, and constraint length n_A, such that d_{min} is the largest integer which satisfies

$$H\left(\frac{d_{min}}{n_A}\right) \leq 1 - R \ ,$$

where $H(x) = -[x\text{Log}_2 x + (1-x)\text{Log}_2(1-x)]$ is the binary entropy function. Since fixed codes can be considered to be periodic codes with period one, this lower bound also applies to the class of periodic codes. This bound is asymptotically the same as Gilbert's lower bound on minimum distance for block codes with the same rate and constraint length [11]. The Gilbert bound is a universally accepted criterion of comparison for codes.

2. Definite Decoding Minimum Distance for Systematic Periodic Codes

Wagner [12], has shown that there exists at least one binary convolutional code with period T = m+1, and rate R, such that

$$H\left(\frac{d_w}{n_w}\right) \geq 1 - 2R,$$

where d_w and n_w are, respectively, different definitions of definite decoding minimum distance and definite decoding constraint length given by Robinson [9]. This bound is only satisfied for $R \leq \frac{1}{2}$. Wagner was the first to derive a Gilbert bound for this class of codes and proved that the bound holds for a class of systematic codes of period $T = (3m+1)/2$ if m is odd or $T = 3m+1$ if m is even. This was the first class of codes investigated in this project and will be further discussed later. This bound has been extended to all rates by Costello and Morrissey [13] to the following: There exists at least one systematic, periodic, time-varying convolutional code such that

$$H\left(\frac{d_{DD}}{n_{DD}}\right) \geq \frac{1-R}{1+R} ,$$

where d_{DD} and n_{DD} are Robinson's definite decoding parameters.

3. Free Distance for Non-systematic Periodic Codes

An obvious lower bound on free distance can be obtained using the following facts concerning minimum distance and column distance:

$$d_{min} = d_m \; ; \; d_{free} = \lim_{j\to\infty} d_j \; ; \; d_{j+1} \geq d_j$$

Therefore, $d_{min} \leq d_{free}$ and all lower bounds on d_{min} are also lower bounds on d_{free}.

Costello [3], p. 81, has proved a lower bound on free distance for non-systematic, periodic codes which is

much tighter than corresponding bounds for fixed codes. His result states that there exists at least one non-systematic periodic code such that

$$\lim_{m\to\infty} \frac{d_{free}}{n_A} \geq \frac{R(1-2^{R-1})}{H(1-2^{R-1}) - R - 1} .$$

This lower bound is very close to the generalized McEliece and Rumsey upper bound for non-systematic fixed codes [3], p. 95. The tightness of this lower bound and its closeness to the upper bound for fixed codes suggests the existence of periodic codes which achieve a much lower probability of decoding error than corresponding fixed codes. It is of interest, therefore, to find such codes and to determine if their superiority over fixed codes is sufficiently great to justify the additional complexity of a time-varying encoder.

III. THE SIMULATED SEQUENTIAL DECODER

Sequential decoding, as discussed in Chapter I, is a method for decoding information which has been encoded, using tree codes, for transmission over a noisy channel. Several sequential decoding algorithms have been introduced and tested. In this chapter a detailed description of the algorithm used in this work will be given along with an explanation of the use of the algorithm in an attempt to construct good time-varying codes.

A. THE DECODING ALGORITHM

The algorithm used, suggested in independent work by Zigangirov [14] and Jelinek [15], is simple to describe and, with modifications proposed by Jelinek, is faster than other competing algorithms. In order to attain the speed advantage without an increase in error probability, the memory required for use by the decoder must be increased considerably. A description of the basic algorithm without modifications follows.

As stated in Section I.D, metric values are assigned to each branch in the code tree. This metric value, given by equation (I.1), is a branch likelihood function. The sum of the branch likelihood functions on a path leading to a particular node is called the node likelihood value. The decoder consists of a portion of computer memory called the stack which is an ordered list of nodes. The stack is

arranged in decreasing order of node likelihood values with the highest node having the greatest value of all nodes in the stack. The stack is first loaded with the value of the origin node which is, thus, the "top" node in the stack. The decoder then proceeds as follows:

(1) Compute the likelihood values of the two successors to the top node and place them into the stack in order determined by their values.

(2) Eliminate the node whose two successors were just added.

(3) If the new "top" node is a terminal node in the tree, stop. Otherwise go to (1).

In other words, after the k^{th} step, the stack will contain exactly k+1 likelihoods corresponding to paths of various lengths and different end nodes. On the $(k+1)^{st}$ step, the decoder searches the stack to find the largest stored likelihood, determines the path to which it corresponds, and replaces that likelihood with the likelihoods of the two successor nodes. When the process halts, the top node in the stack determines a path from the origin node to a terminal node in the tree.

Geist [16], p. 38, has shown that the Zigangirov-Jelinek algorithm chooses a path $s_0, s_1, \ldots, s_L$ through the tree which satisfies the following conditions:

(1) s_0 is the origin node.

(2) The branch leading from any particular node, s_i, to a node in the next level of the tree is the first branch of one of the paths from s_i to the end of the tree having the greatest minimum likelihood value.

In essence, the result of decoding is such that the decoder picks a node, checks all paths from that node to

the end of the tree to find the one with greatest minimum value, and "moves" one node farther along that path. The process repeats until a terminal node is reached. To meet the essential feature of sequential decoding given by Jacobs and Berlekamp [6], the branches must be examined sequentially, so that at any node of the tree the decoder's choice among a set of previously unexplored branches may not depend on received branches deeper in the tree. Therefore, the decoder does not in actuality look ahead on all possible paths, but selects for a node to be extended, the one with greatest likelihood value from the set of all nodes in the stack.

Since the decoder performs one computation for each node it examines, and with non-zero probability the number of nodes examined grows quite large, there is a possibility of stack overflow. In simulation studies this was handled by a cessation of decoding of the particular frame during which overflow occurred and starting the next frame. The frame which caused overflow was then counted as an erasure.

A problem with the Zigangirov-Jelinek algorithm is the excessive time required to keep the stack ordered so that the nodes are in decreasing order of likelihood values. This problem is alleviated when the algorithm is modified as proposed by Jelinek [15]. The modified version, now to be discussed, is the one which was used in the experimental phase of this work.

The modification consists of quantizing the likelihood values and placing nodes in ordered bins according to their

values. The stack is then arranged as random access storage with descriptions of each node examined placed into the stack sequentially from the top. Decoding proceeds as follows:

(1) Select a node from the highest non-empty bin, compute the values of its successors, and place them in the proper bins.

(2) Eliminate from further consideration the node whose successors were just added.

(3) If a node in the highest non-empty bin is a terminal node in the tree, stop. Otherwise, go to (1).

In each case when a node is placed in a bin, its description is placed on the stack.

Geist [17] has done extensive computer simulation studies which indicate that this version of the Jelinek algorithm is, in most cases of interest, superior to the Fano algorithm [18] as long as sufficient computer memory is available. A detailed discussion of the process followed in the decoder program is given in Appendix A.

B. USE OF THE DECODER FOR CONSTRUCTING CODES

As was stated in Chapter II, the error-correcting capabilities of a code are closely related to the respective weights of the codewords. Forney [19] has given a method for determining the codeword weights by using a sequential decoding simulator. His method is outlined for a simulator using the Fano algorithm for determining the codeword weights of fixed codes. In this project the method was suitably modified to use the quantized Jelinek algorithm for constructing time-varying codes with good distance properties.

Since with time-varying codes, free distance appears to be the most appropriate distance measure, and free distance is related to order-j column distance, the decoder was used to measure the column distance of successive codewords. With this value available, an attempt was then made to select subsequent generator polynomials with the aim of maximizing the minimum order-j column distance.

The steps in the procedure are as follows:

(1) Select the first generator polynomials, $\underline{G}_0^{(k)}$, such that $d_1 = 2$. Set $i = 1$, $j = 2$.

Note: $k = 1$ for systematic codes.
$k = 1,2$ for non-systematic codes.

(2) Make a choice for $\underline{G}_i^{(k)}$.

(3) Using these generator polynomials and the decoder, search to level j in the tree.

(4) Compute d_j. If $d_j > d_{j-1}$ go to (7).

(5) If all choices for $\underline{G}_i^{(k)}$ are exhausted, go to (7).

(6) Make another choice for $\underline{G}_i^{(k)}$. Go to (3).

(7) If $i = T$, the period of the code, stop. Otherwise, $j = j+1$, $i = i+1$, and go to (2).

After this process was completed, it was a simple matter to modify the computer program to test the code thus generated.

IV. CONSTRUCTION OF TIME-VARYING CODES

In this chapter, the several methods employed in an attempt to construct good time-varying codes will be discussed. Included will be the reasons for searching for a code of the particular class, and a comparison to the performance of fixed codes of the same memory. The distribution of computations, number of frames erased per 1000 frames, number of error frames per 1000 frames, and bit error probability are compared for each code generated.

All codes generated were of rate ½ and comparisons were made over simulated binary symmetric channels with crossover probabilities $p = 0.03125$ and $p = 0.04688$. For each code, an attempt was made to decode 1000 frames of length 256 bits at each of the two channel probabilities. Throughout simulation, the all-zero sequence was assumed to have been transmitted. Since the nominal probability of a transmitted symbol being a zero or a one is one-half, no loss of generality occurs, but this affords the decoder an unfair advantage. For this reason, the decoder was programmed to choose the "one-branch" in cases where ties occurred.

A. SYSTEMATIC TIME-VARYING CODES

1. Wagner Codes

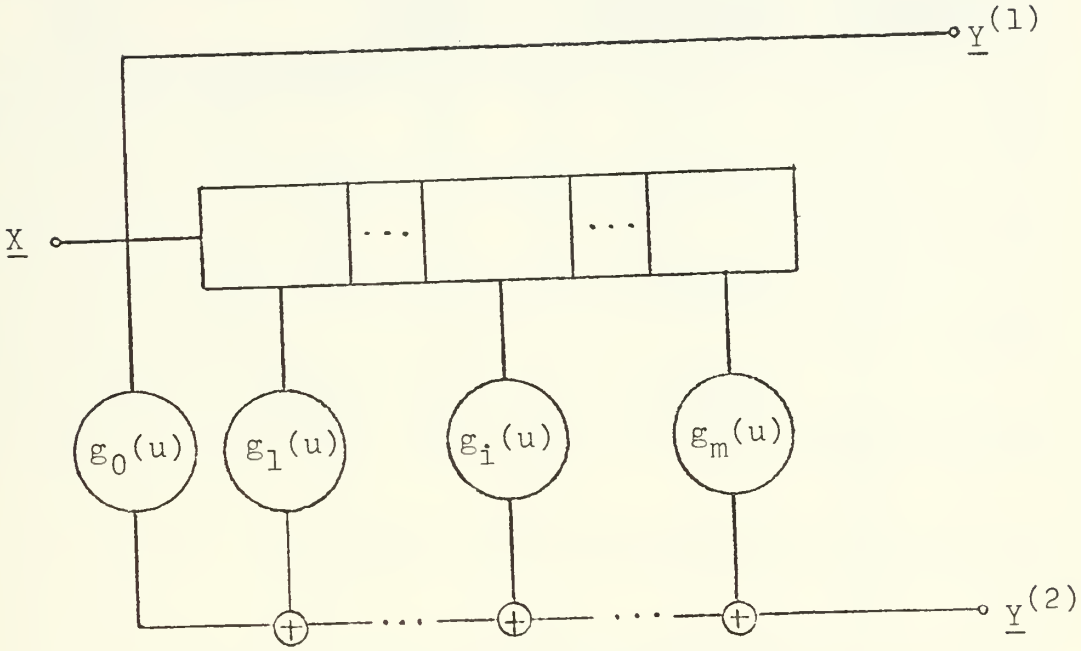
Wagner [12] suggested a class of systematic, periodic, binary convolutional codes for which his Gilbert

bound applies. The encoder is depicted in Figure 5(a). As before, if $g_i(u) = 0$, no wire is present, and if $g_i(u) = 1$, a wire is present. The generator sequence, $g_0(u),\ldots,g_m(u)$, is formed from the leftmost (m+1) bits of the generator register shown in Figure 5(b). Each time unit, the information register shifts right once and the generator register shifts circularly twice. The period of the code is $T = 3m+1$ if m is even and $T = (3m+1)/2$ if m is odd.

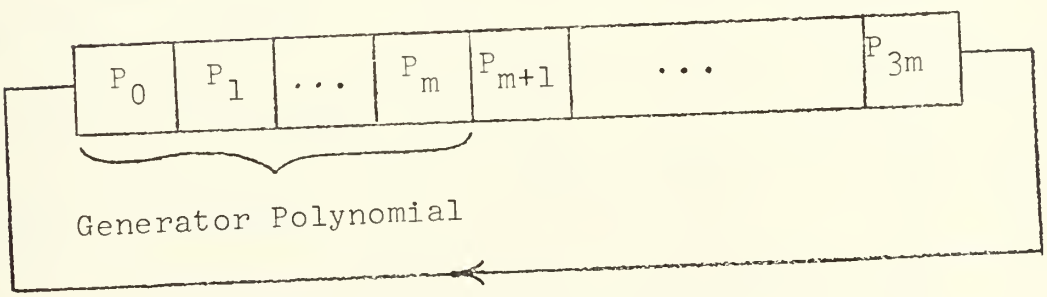
Using the fact that the output sequence from a k-stage maximal length feedback shift register is periodic with period 2^k-1, a five-stage MLFSR was used to generate codes of m = 10 with T = 31. A complete discussion of feedback shift registers is contained in Peterson [20], p. 147. To generate the code, coefficients of all primitive polynomials of degree five were tried as the pattern of taps on the MLFSR. The code generated which gave the best performance is compared in Table II to the performance of a fixed systematic code constructed by Costello [3] for high free distance. Costello's code was constructed for m = 35 (36 bits) and taking the first 11 bits to form a memory ten code actually resulted in a generator polynomial with m = 9. The truncated version (in octal form) is:

$$\underline{G} = 7324.$$

Two modifications of the generated Wagner code were tried in an effort to improve performance. In the first modification, called "Wagner (Mod I)", the code was generated as before with the exception that it was insured that



(a). Time-varying Encoder.



(b). Generator Register.

Figure 5. Convolutional Encoder for Wagner Codes.

CODE	ERROR FRAMES	BIT ERROR PROBABILITY	ERASED FRAMES
Wagner (p = 0.03125)	200	0.00200	0
Costello (p = 0.03125)	72	0.00101	0
Wagner (p = 0.04688)	495	0.00764	0
Costello (p = 0.04688)	323	0.00614	1

(a). Errors and Erasures.

N	300	500	800	1000
Wagner (p = 0.03125)	638	11	2	0
Costello (p = 0.03125)	252	2	1	0
Wagner (p = 0.04688)	503	55	3	0
Costello (p = 0.04688)	622	19	3	2

(b). Frames with Number of Computations > N.

Table II. Comparison of Wagner Code to Fixed Systematic Code (m = 10).

the present information bit was included in the parity-check sequence (i.e., $g_0(u) = 1$ for all u). The results compared to Costello's code are given in Table III.

This modification gave only a small improvement over the true Wagner code with the bit error probability at a channel error probability of p = 0.04688 being slightly less than that of Costello's code.

In the second modified version, the memory of the encoder was increased to 12 and $g_2(u),\ldots,g_{12}(u)$ were generated as for the true Wagner code. It was then insured that the present information bit and one past information bit were included in the parity-check sequence (i.e., $g_0(u) = g_1(u) = 1$ for all u). This code was called "Wagner (Mod II)" and Table IV shows a comparison of results obtained to those obtained using Costello's code with m = 12. The fixed code generator polynomial (in octal form) is:

$\underline{G} = 73244.$

As expected, this code was an improvement over the true Wagner code since the memory was longer, but the code did not show improvement over a comparable fixed code.

2. Shift Register Codes

These codes were generated using a 24-stage maximal length feedback shift register. The first, called "Code A," was formed by taking as the generator polynomials each successive 24-bit output sequence obtained after shifting the register right two bits. The second, Called "Code B," was formed in the same manner except that the output

CODE	ERROR FRAMES	BIT ERROR PROBABILITY	ERASED FRAMES
Wagner (Mod I) (p = 0.03125)	134	0.00142	0
Costello (p = 0.03125)	72	0.00101	0
Wagner (Mod I) (p = 0.04688)	397	0.00604	0
Costello (p = 0.04688)	323	0.00614	1

(a). Errors and Erasures.

N	300	500	800	1000
Wagner (Mod I) (p = 0.03125)	516	7	1	0
Costello (p = 0.03125)	252	2	1	0
Wagner (Mod I) (p = 0.04688)	601	34	2	1
Costello (p = 0.04688)	622	19	3	2

(b). Frames with Number of Computations > N.

Table III. Comparison of Wagner (Mod I) Code to Fixed Systematic Code (m = 10).

CODE	ERROR FRAMES	BIT ERROR PROBABILITY	ERASED FRAMES
Wagner (Mod II) (p = 0.03125)	64	0.00093	0
Costello (p = 0.03125)	28	0.00051	0
Wagner (Mod II) (p = 0.04688)	271	0.00486	0
Costello (p = 0.04688)	179	0.00378	0

(a). Errors and Erasures.

N	300	500	800	1000
Wagner (Mod II) (p = 0.03125)	354	12	2	0
Costello (p = 0.03125)	318	9	2	1
Wagner (Mod II) (p = 0.04688)	676	54	3	0
Costello (p = 0.04688)	766	41	2	1

(b). Frames with Number of Computations > N.

Table IV. Comparison of Wagner (Mod II) Code to Fixed Systematic Code (m = 12).

sequences were taken after shifting the register left one bit. For performance comparison, Costello's fixed systematic code with m = 23 was used:

$$\underline{G} = 73246070.$$

Again, performing the truncation on the m = 35 code yielded only a code with memory m = 20. Code A was chosen since the generator polynomials vary like those of a Wagner code although the period of the code is much greater than (3m+1)/2. Code B is a modification which was tested for comparison. Results are given in Table V.

3. High Minimum Column Distance Codes

This class of codes was generated using the algorithm outlined in Section II.B. The algorithm is a modification of Costello's algorithm A1 [3], p. 114, for constructing rate ½, fixed, systematic codes with high free distance. The difficulty with trying to construct time-varying codes in this manner is brought about by the fact that when deciding upon the generator polynomial, $\underline{G}(u)$, none of the polynomials, $\underline{G}(u+t)$, $1 \leq t < T-u$, are known. Therefore, one cannot measure the order-j column distance, d_j, for $j > u$ to get an estimate of the magnitude of d_{free} for the polynomials chosen up to that point. The codes were constructed with the aim of maximizing d_j for each step, keeping in mind that this method only gives high minimum column distance measured from the origin node and that the minimum column distance measured from other nodes may be quite low. The codes were constructed for a high d_{35}. Comparison to

CODE	ERROR FRAMES	BIT ERROR PROBABILITY	ERASED FRAMES
Code A (p = 0.03125)	18	0.00032	1
Code B (p = 0.03125)	20	0.00023	3
Costello (p = 0.03125)	6	0.00014	0
Code A (p = 0.04688)	91	0.00177	28
Code B (p = 0.04688)	104	0.00174	34
Costello (p = 0.04688)	40	0.00122	12

(a). Errors and Erasures

N	300	500	800	1000
Code A (p = 0.03125)	939	104	22	16
Code B (p = 0.03125)	939	122	27	17
Costello (p = 0.03125)	596	23	4	3
Code A (p = 0.04688)	909	398	161	107
Code B (p = 0.04688)	895	397	160	117
Costello (p = 0.04688)	905	163	60	39

(b). Frames with Number of Computations > N.

Table V. Comparison of Systematic Shift Register Codes to Fixed Systematic Code (m = 23).

fixed code performance is given in Table VI. The codes were generated for m = 23 with period T = 35. Costello's code was used for comparison.

B. NON-SYSTEMATIC TIME-VARYING CODES

As was discussed in Chapter II, Costello [3] derived a lower bound on free distance for non-systematic, periodic, convolutional codes which suggests that there exists at least one code of this class with decoding properties offering a great improvement over comparable fixed codes. For this reason, a search for members of this class of codes was conducted.

1. Wagner-like Codes

Since Wagner [12] was the first to offer any results concerning periodic codes, a non-systematic version of his codes was tested. Both arrays of generator polynomials were constructed using the method outlined in Section IV.A.1 for m = 10. The performance results were compared to a non-systematic, high free distance, fixed code constructed by Costello with his algorithm A9 [3], p. 140. Costello's code, again, was constructed for m = 35 and the generator polynomials (in octal form), truncated to m = 10 are:

$$\underline{G}^{(1)} = 7334$$

$$\underline{G}^{(2)} = 5334$$

As before, truncation actually resulted in a code with m = 9. Performance results are given in Table VII.

CODE	ERROR FRAMES	BIT ERROR PROBABILITY	ERASED FRAMES
High d_{35} (p = 0.03125)	53	0.00075	4
Costello (p = 0.03125)	6	0.00014	0
High d_{35} (p = 0.04688)	210	0.00388	56
Costello (p = 0.04688)	40	0.00122	12

(a). Errors and Erasures.

N	300	500	800	1000
High d_{35} (p = 0.03125)	902	308	89	44
Costello (p = 0.03125)	596	23	4	3
High d_{35} (p = 0.04688)	787	489	263	181
Costello (p = 0.04688)	905	163	60	39

(b). Frames with Number of Computations > N.

Figure VI. Comparison of High d_{35} Code to Fixed Systematic Code (m = 23).

CODE	ERROR FRAMES	BIT ERROR PROBABILITY	ERASED FRAMES
Wagner (p = 0.03125)	21	0.00042	8
Costello (p = 0.03125)	13	0.00029	1
Wagner (p = 0.04688)	77	0.00303	117
Costello (p = 0.04688)	54	0.00186	15

(a). Errors and Erasures.

N	300	500	800	1000
Wagner (p = 0.03125)	978	441	104	57
Costello (p = 0.03125)	343	24	6	4
Wagner (p = 0.04688)	923	805	450	324
Costello (p = 0.04688)	893	181	81	50

(b). Frames with Number of Computations > N.

Table VII. Comparison of Non-systematic Wagner-like Codes to Fixed Non-systematic Codes (m = 10).

2. Shift Register Codes

This m = 23 code was generated in like manner to the systematic code of the same type (Code A) with the exception that a different 24-stage MLFSR was used to form each array of generator polynomials. Comparison of performance is made in Table VIII and Figures 6 and 7 to Costello's non-systematic fixed code, truncated to m = 23. The generator polynomials for the fixed comparison code are:

$$\underline{G}^{(1)} = 73353367$$

$$\underline{G}^{(2)} = 53353367$$

Since no errors were made with either the time-varying code or the fixed code, the dominant parameter for comparison is the distribution of computations. Therefore, plots of the distribution are given vice tables. As was stated in Chapter I, the computation has an assymptotic Pareto distribution of the form

$$P\{C \geq N\} = KN^{-\rho} .$$

The observed distributions for these codes are plotted in Figures 6 and 7 on a log-log scale. Therefore, the curves should approach straight lines whose slopes are the negatives of the values of ρ given in Table I. These asymptotes are displayed in the figures.

From Figures 6 and 7 it can be seen that, although the shift register code made no decoding errors, the code caused the decoder to perform many more computations than it did with the fixed code.

CODE	ERROR FRAMES	BIT ERROR PROBABILITY	ERASED FRAMES
Shift Register (p = 0.03125)	0	0.0	14
Costello (p = 0.03125)	0	0.0	2
Shift Register (p = 0.04688)	0	0.0	133
Costello (p = 0.04688)	0	0.0	34

Table VIII. Comparison of Non-systematic Shift Register Codes to Fixed Non-systematic Code (m = 23).

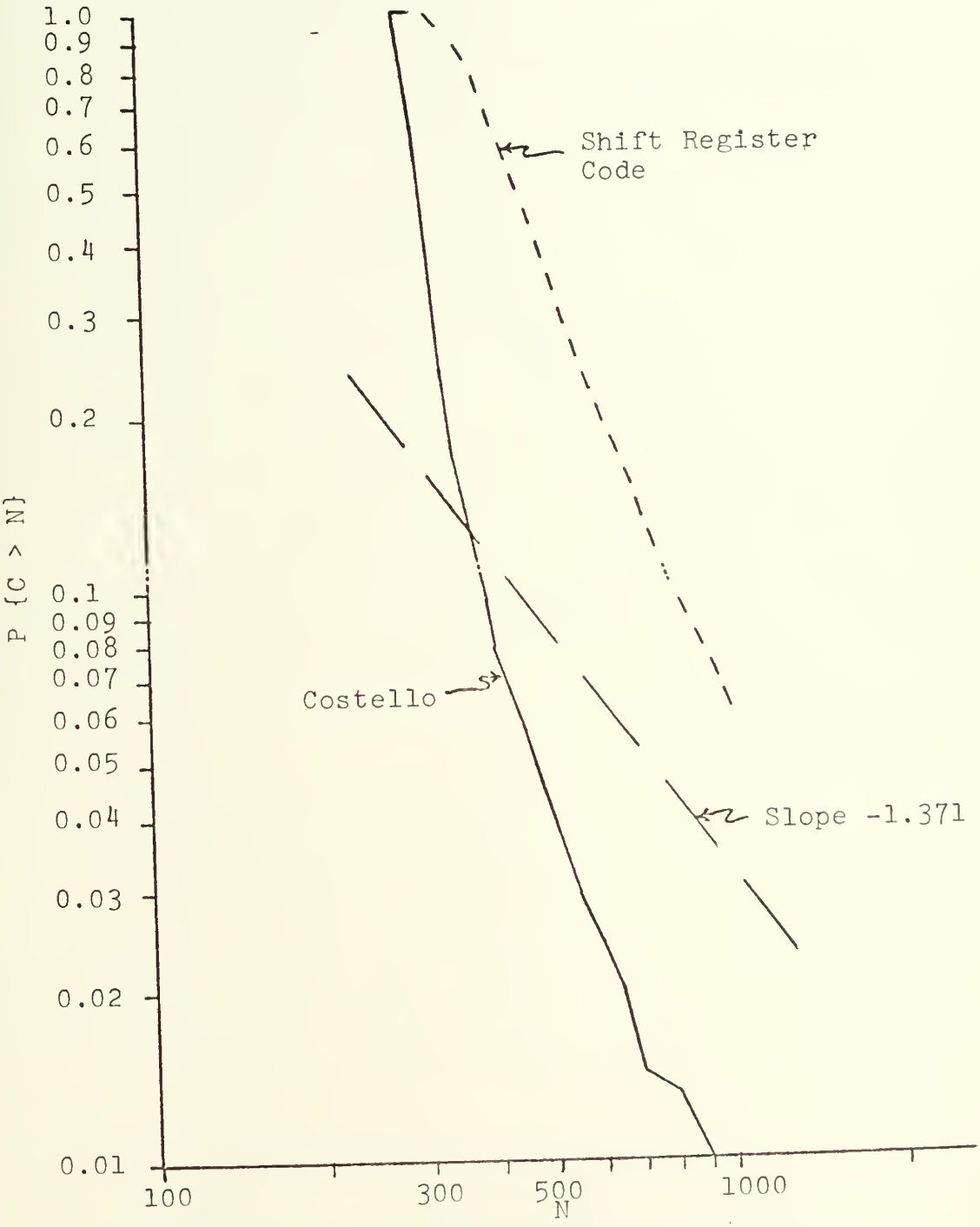


Figure 6. Distribution of Computations Non-systematic Codes (m = 23, p = 0.03125).

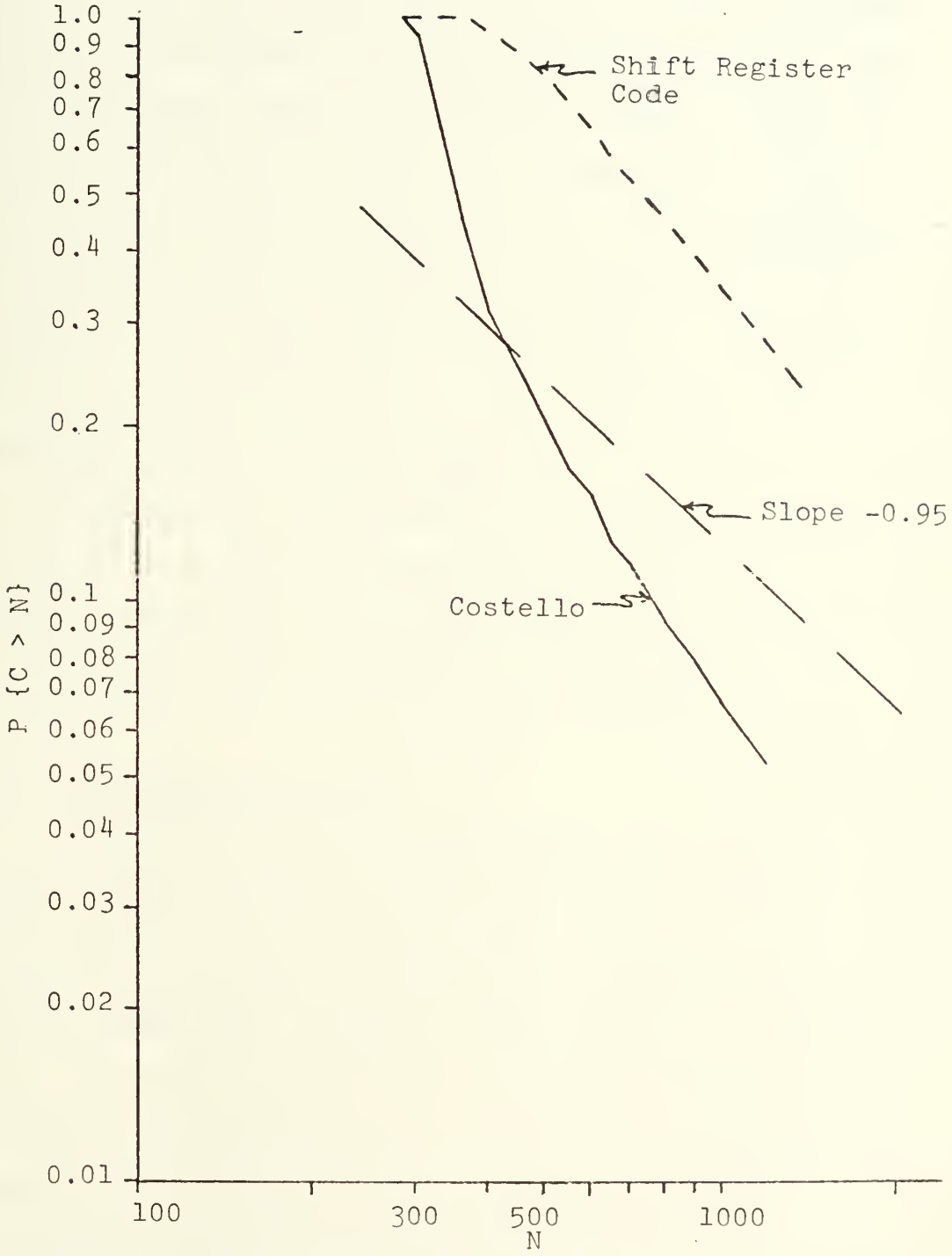


Figure 7. Distribution of Computations Non-systematic Codes (m = 23, p = 0.04688).

3. High Minimum Column Distance Codes

Non-systematic, time-varying, high minimum column distance codes were next investigated. These codes were generated using the algorithm in Section II.B to construct two arrays of generator polynomials with a high d_{35}. The same difficulties noted for systematic codes of this type also pertain to the generation of non-systematic codes. Again, codes of m = 23 with period T = 35 were generated and Costello's high free distance, non-systematic code truncated to m = 23 was used for comparison. The results are given in Table IX. Plots of the distribution of computations are given in Figures 8 and 9.

Although the probability of a decoding error is quite low with this time-varying code, the number of computations performed when using the code is much greater than the number performed when using a comparable fixed code.

4. Complementary Codes

Recently, Bahl and Jelinek [21] have investigated a class of rate ½ fixed convolutional codes with complementary generators. They have described a synthesis and a search procedure that results in good codes up to memory m = 23. The codes constructed come close to achieving established upper bounds on free distance and the generator polynomials satisfy the following two conditions:

(1) $g_0^{(1)} = g_0^{(2)} = g_m^{(1)} = g_m^{(2)} = 1$

(2) $g_i^{(2)} = g_i^{(1)*}, \qquad 0 < i < m,$

CODE	ERROR FRAMES	BIT ERROR PROBABILITY	ERASED FRAMES
High d_{35} (p = 0.03125)	3	0.00003	7
Costello (p = 0.03125)	0	0.0	2
High d_{35} (p = 0.04688)	3	0.00002	110
Costello (p = 0.04688)	0	0.0	34

(a). Errors and Erasures.

N	300	500	800	1000
High d_{35} (p = 0.03125)	762	133	63	40
Costello (p = 0.03125)	637	39	13	7
High d_{35} (p = 0.04688)	975	463	296	244
Costello (p = 0.04688)	947	219	107	80

(b). Frames with Number of Computations > N.

Table IX. Comparison of Non-systematic High d_{35} Code to Fixed Non-systematic Code (m = 23).

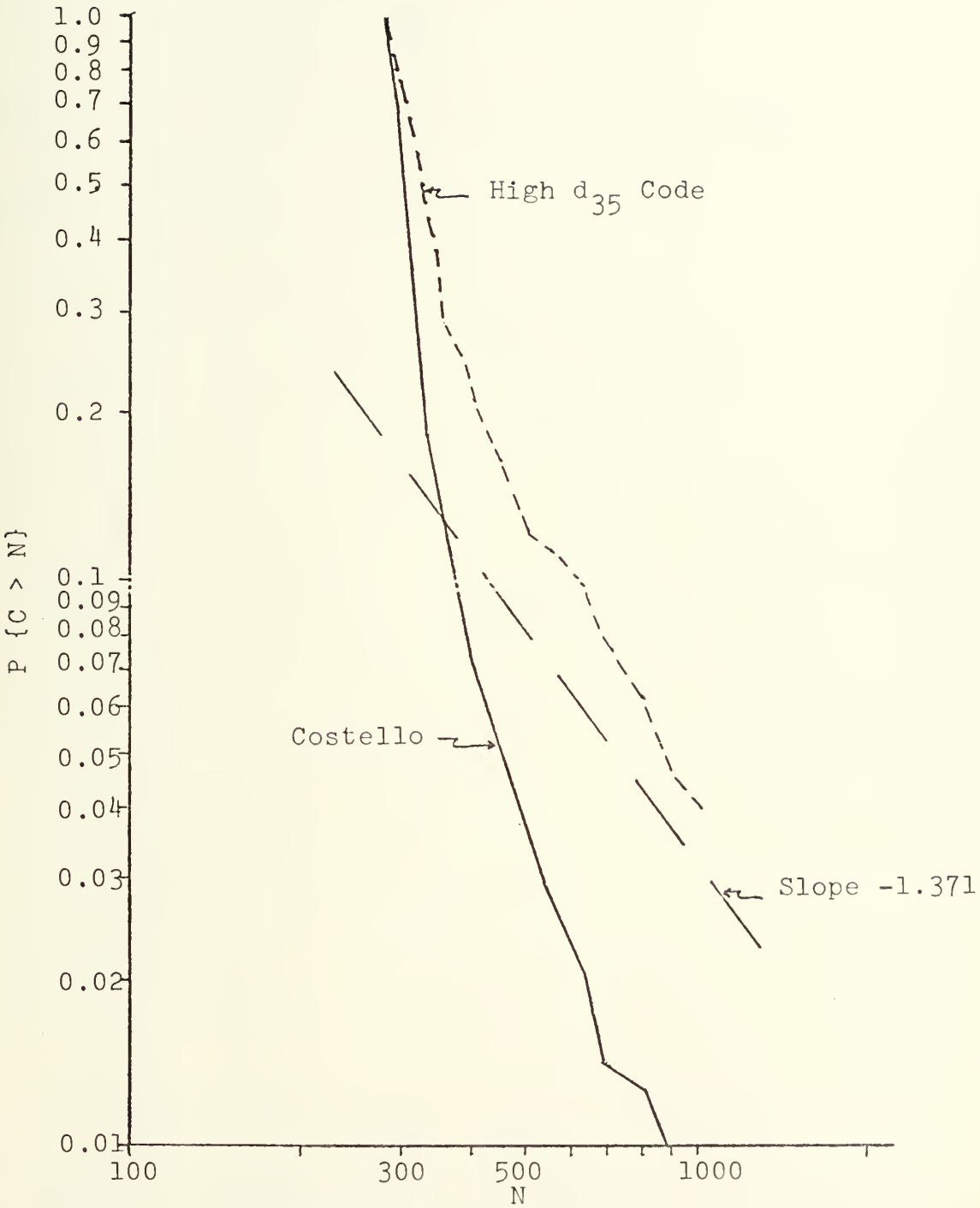


Figure 8. Distribution of Computations High d_{35} Non-systematic Codes (m = 23, p = 0.03125).

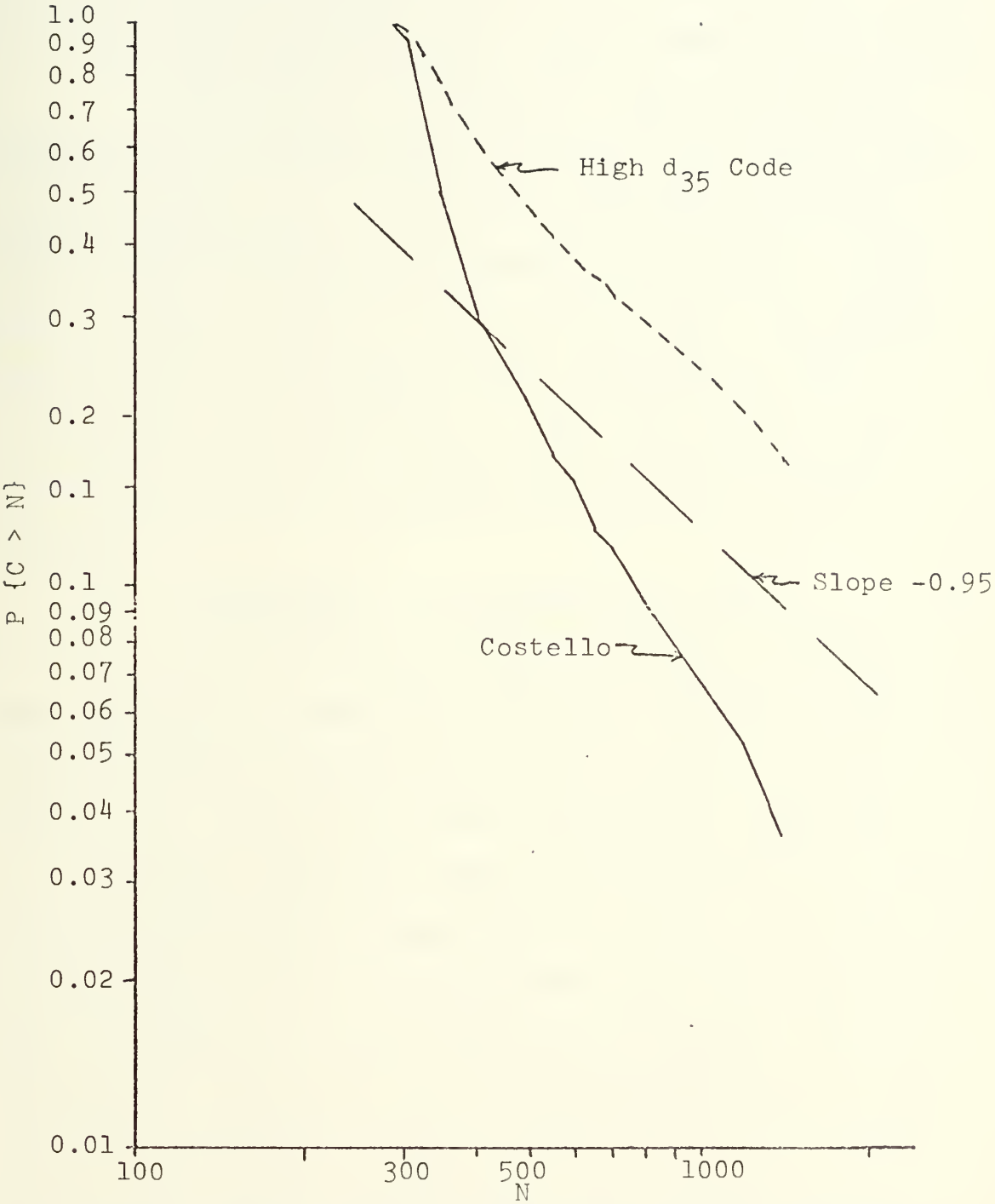


Figure 9. Distribution of Computations High d_{35} Non-systematic Codes (m = 23, p = 0.04688).

where $g_i^{(1)*}$ denotes the complement of $g_i^{(1)}$. The results of this report [21] suggested a search for periodic codes with complementary generators.

a. High Minimum Column Distance Complementary Codes

Construction of codes with high d_{35} was again attempted using the algorithm of Section II.B ensuring that the choices made for the generator polynomials satisfied the two stated conditions for complementary codes. Codes with m = 23 and period T = 35 were generated. Performance results are given in Table X with a comparison to the m = 23 fixed code constructed by Bahl and Jelinek. Plots of the distribution of computations are given in Figures 10 and 11.

b. Alternating Two Complementary Codes

In this time-varying code, which is the last type investigated, generator arrays were formed by alternating two fixed complementary codes to give a code with period T = 2. Since the fixed complementary codes with m = 23 used for comparison make no decoding errors, it was decided to use shorter memory codes which do cause some decoding errors so as to have a good basis for comparison. Bahl and Jelinek [21] have constructed a number of optimal complementary codes that have the largest attainable free distance. The complete list of this set for $2 \leq m \leq 8$ is given in the report. Two of these codes with m = 6 were chosen to form the period T = 2 code. The four generator polynomials (in octal form) are:

CODE	ERROR FRAMES	BIT ERROR PROBABILITY	ERASED FRAMES
Periodic Complementary (p = 0.03125)	7	0.00010	4
Fixed Complementary (p = 0.03125)	0	0.0	1
Periodic Complementary (p = 0.04688)	7	0.00011	66
Fixed Complementary (p = 0.04688)	0	0.0	27

(a). Errors and Erasures

N	300	500	800	1000
Periodic Complementary (p = 0.03125)	665	72	28	21
Fixed Complementary (p = 0.03125)	619	31	11	7
Periodic Complementary (p = 0.04688)	960	331	178	137
Fixed Complementary (p = 0.04688)	948	190	77	67

(b). Frames with Number of Computations > N.

Table X. Comparison of Periodic Complementary Codes to Fixed Complementary Codes (m = 23).

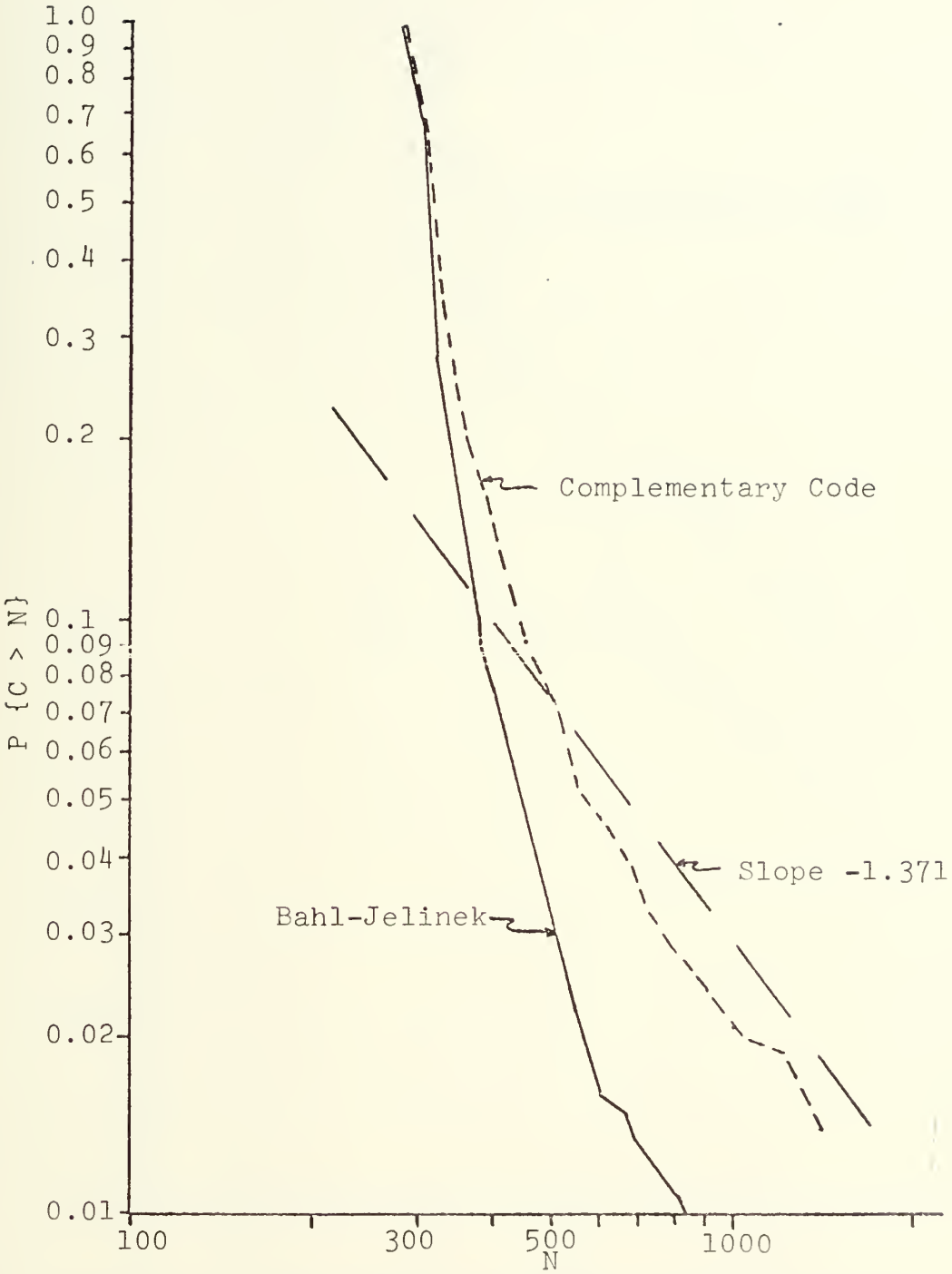


Figure 10. Distribution of Computations Periodic Complementary Codes (m = 23, p = 0.03125).

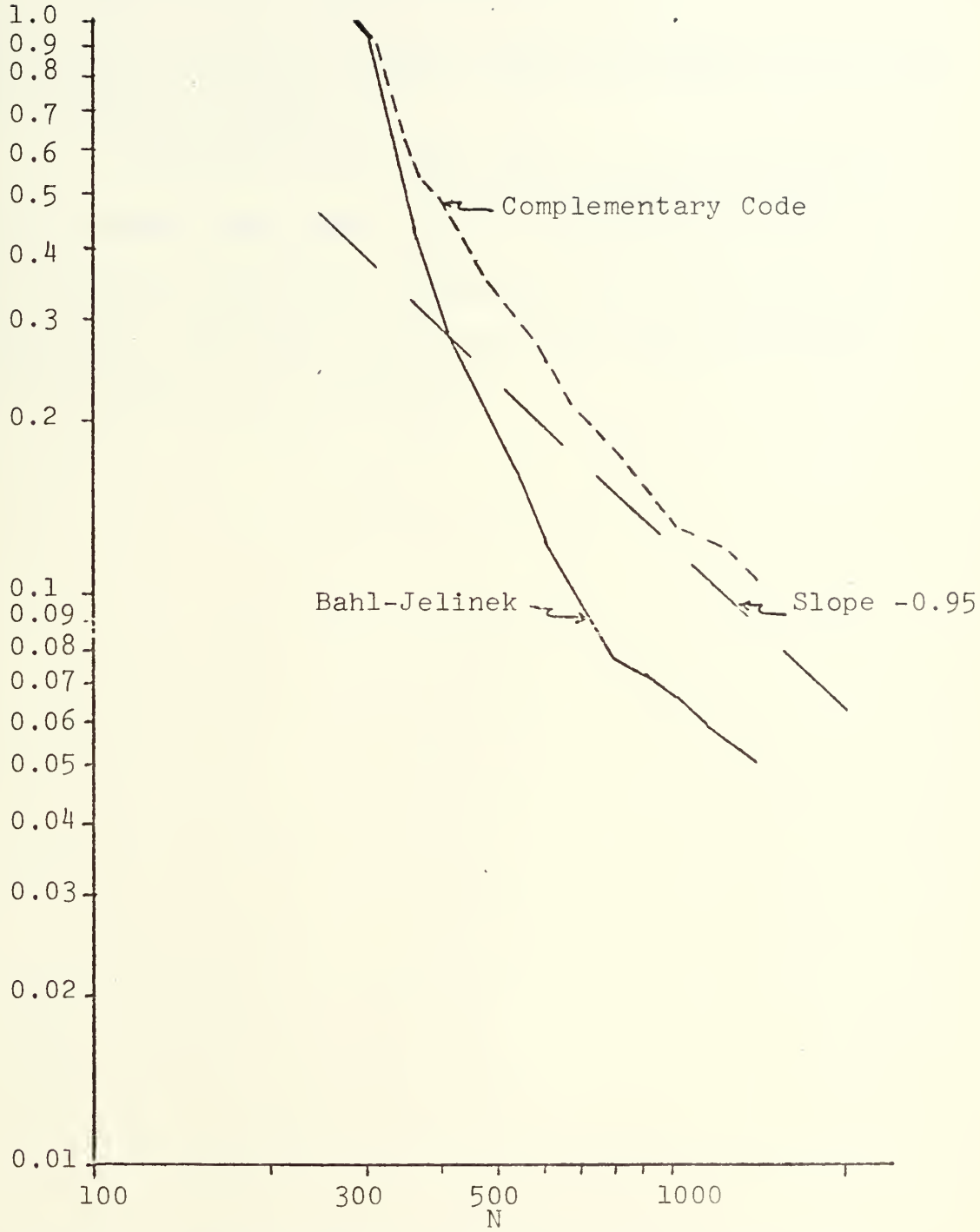


Figure 11. Distribution of Computations Periodic Complementary Codes (m = 23, p = 0.04688).

$$\underline{G}_1^{(1)} = 424 \qquad \underline{G}_1^{(2)} = 754$$

$$\underline{G}_2^{(1)} = 514 \qquad \underline{G}_2^{(2)} = 664$$

Performance results for the two fixed codes and the periodic code are given in Table XI.

As can be seen from Table XI, the performance of the periodic code comes close to equaling that of the fixed codes in both error probability and distribution of computations, but major improvement over the performance of fixed codes is not exhibited.

CODE	ERROR FRAMES	BIT ERROR PROBABILITY	ERASED FRAMES
Periodic (p = 0.03125)	7	0.00020	1
Fixed #1 (p = 0.03125)	8	0.00020	1
Fixed #2 (p = 0.03125)	13	0.00027	1
Periodic (p = 0.04688)	55	0.00231	14
Fixed #1 (p = 0.04688)	46	0.00138	9
Fixed #2 (p = 0.04688)	58	0.00188	8

(a). Errors and Erasures.

N	300	500	800	1000
Periodic (p = 0.03125)	369	37	5	3
Fixed #1 (p = 0.03125)	413	36	8	3
Fixed #2 (p = 0.03125)	346	21	5	2
Periodic (p = 0.04688)	892	161	55	35
Fixed #1 (p = 0.04688)	903	185	57	39
Fixed #2 (p = 0.04688)	890	162	59	33

(b). Frames with Number of Computations > N.

Table XI. Comparison of Period-2 Complementary Codes to Fixed Complementary Codes (m = 6).

V. CONCLUSIONS

In all types of time-varying codes investigated, the results obtained did not show improvement over the performance of comparable fixed codes. The most promising periodic codes appear to be those using complementary generators with the generator arrays formed by some manner of interleaving optimum fixed code polynomials. Of note here is the fact that the Bahl-Jelinek fixed convolutional codes with complementary generators exhibited better performance than other fixed codes with longer memory.

Further research into time-varying codes should include a study of the distance properties of good codes with the aim of finding a systematic method to generate periodic codes with high free distance. Experimental methods might entail performing perturbations on the generator arrays of the best known periodic code in an effort to improve performance.

Costello's lower bound on free distance for rate ½, non-systematic, periodic codes [3], p. 81, is an asymptotic bound which means that it may not necessarily apply to codes of short constraint length. This fact is exemplified by the results obtained in this work with periodic codes up to memory m = 23. The added encoder/decoder complexity required to generate long constraint length time-varying codes is not practical when one considers that equally

reliable communications can be achieved using less complex, shorter fixed codes. Time-varying codes, thus far, appear to be of theoretical significance only.

APPENDIX A

DECODER AND ERROR PATTERN GENERATOR PROGRAMS

All computer simulation was accomplished on the XDS-9300 computer. A detailed discussion of the quantized Jelinek decoder and the error pattern generator programs will now be given.

A. DECODER PROGRAM

For every node the decoder encounters in its search through the tree, three items of information about the node must be saved so that the node can be extended if it becomes the top node in the stack. These items are the node likelihood value, its depth into the tree, and the encoder state at the node. The encoder state is necessary so that the code symbols on the two branches emanating from the node may be determined; the depth is necessary so that these code symbols may be compared with the corresponding symbols in the received sequence; and the new node likelihood values must be obtained by adding the branch likelihood values of the two emanating branches to the extended node value.

For the node to be extended, these items were saved in three computer storage locations with the symbolic labels VALUE, DEPTH, and STATE respectively. For each node on the stack, six consecutive words of computer memory contained the complete node description. In the first three of these were stored the node value, depth, and encoder state; the

last three locations contained a stack pointer, a path pointer, and a flag, all of which will be explained later.

During simulation it was convenient to have all positive node values; so the origin node was biased to a value of 1000 which ensured that no node encountered had a negative value.

At the beginning of the search the origin node is the top node. Therefore, DEPTH and STATE are initially set to zero, and VALUE is set to 1000. During the search, the information about each node encountered is saved either in VALUE, DEPTH, and STATE, or as a node description on the stack (or both). Since, in this quantized version of the algorithm, node descriptions are neither rearranged in the stack nor physically deleted from the stack, even if the node reaches the top of the stack and is extended, there are several node descriptions in memory which are no longer on the stack. Therefore, a method is necessary whereby the ordering of the stack contents into bins may be accomplished and node descriptions which represent nodes still on the stack may be determined. The aforementioned stack pointer and two arrays called bin index and bin inventory are used for this purpose. For each bin, the bin inventory contains the number of nodes in the bin, and the corresponding location in the bin index contains the address of the first word of the node description for one of the nodes in the bin. The stack pointer for this node contains the address of the first word of the node description for another node in the bin. The chain continues until all nodes in the bin

are linked together with the stack pointer of the node description of the last node in the bin being empty.

The size of the bin index and bin inventory arrays are determined by the size of the node likelihood value quantizing increment which was assigned the symbolic label H, and which was chosen so as to satisfy the following two conditions:

(1) H is at least as great as the maximum positive branch likelihood value, normalized to an integer.

(2) H is restricted to be a power of two (i.e., $H = 2^r$) so that the proper bin in which to place a node can be determined by an r-bit shift operation.

To find the value for H which satisfies the first condition, consider that when a node is extended and the two branches are compared to the received sequence, there are three possible outcomes: (1) a branch can agree with the received sequence in both digits; (2) a branch can disagree with the received sequence in both digits; (3) a branch can disagree in exactly one digit. The branch likelihood values correspond to the branch metric values in Equation (I.1), normalized to integer values. For the three possible outcomes, the metric values at each of the two channel probabilities were chosen as given in Table XII. From the information listed in Table XII, it can be seen that a quantizing increment which satisfied both stated conditions, and the one used during simulation, was H = 8.

The sequence of events during decoding will now be described. VALUE, DEPTH, and STATE are set to the respective parameters for the top node in the stack and the

	Metric Values	
	p = 0.03125	p = 0.04688
Both digits agree	8	8
Both digits disagree	-72	-64
One digit disagrees	-32	-28

Table XII. Metric Values Used in Simulation.

branch likelihood values for the two branches emanating from that node are computed. Now there are two possible cases: either (1) at least one branch value is positive, or (2) both branch values are negative. The two cases will be considered separately.

Case (1). If at least one branch value is positive, the successor node along that path must belong in either the same bin as the extended node or a higher bin and, therefore, the successor node must be the new top node. VALUE, DEPTH, and STATE are adjusted to the corresponding parameters for the new node and the node description for the other successor must be stored on the stack. The first three items are computed from the parameters stored in VALUE, DEPTH, and STATE along with the calculated branch value. The integer part of the successor node value divided by H determines the proper bin in which the node belongs. The bin inventory for this bin is increased by one, and the stack pointer of the new node description is set to the

address which is stored in the corresponding location of the bin index. The bin index is then set to the address of the first word of the new node description. The path pointer and the flag, which will be discussed later, are used to recover the information symbols on the chosen path after a search ends on a terminal node in the tree. They are now set and the decoder is ready to extend the new top node using the present parameters in VALUE, DEPTH, and STATE.

Case (2). If both branch values are negative, it is clear that there may be nodes on the stack which have greater likelihood values than the two successors, and therefore belong in higher bins. First the two new node descriptions are stored on the stack. Now, since the extended node was in the highest non-empty bin, the decoder searches down the bin inventory array, starting with the bin in which the extended node was located, until it finds a bin with non-zero inventory. When this bin is located, the inventory is decreased by one, VALUE, DEPTH, and STATE are set to the respective parameters from the node description located using the address in the corresponding bin index, and the bin index is set to the stack pointer address for that node. The decoder is now ready to extend the new node.

As was stated earlier, each frame is 256 bits long which yields a 256 branch tree. The 256 branches are followed by an m-branch tail corresponding to a memory span of zeros. Since it is known that only zeros are transmitted in the

tail, the decoder considers only the zero-branch successor during its search in the tail. If the branch value is positive, no node description is stored and the successor node is extended. If the branch value is negative, the successor node description is stored on the stack and a new top node is located in the same manner as described in Case (2) earlier.

When the contents of DEPTH is (256 + m), the search has ended on a terminal node and the decoder must recover the information symbols on the chosen path. The last two items of the node description are used to accomplish this function. Had a description of every node encountered been stored, the path pointer could have been set to the address of the description of its predecessor and a path from the terminal node to the origin would easily have been determined; but this is not the case, since, in many instances, only one node description is stored. The path pointer is set to the address of the description of a node's predecessor if it is stored, or, to the address of the node at the end of the second branch emanating from the predecessor if it is not stored. In the latter case the flag is used to indicate whether the path pointer leads to the predecessor or the second branch. The decoder can then follow a path back to the origin to retrieve the information symbols along the path selected.

B. ERROR PATTERN GENERATOR PROGRAMS

1. Binary Symmetric Channel (p = 0.03125)

The binary symmetric channel with crossover probability 0.03125 = 1/32 was simulated using a 23-stage maximal-length feedback shift register. The periodic sequences from a MLFSR are called "pseudo-noise" (or p-n) sequences. If a sequence of i digits ($i \leq m$) is picked at random from the output sequence of an m-stage binary MLFSR, then each non-zero sequence has probability $2^{m-i}/(2^m-1)$ and the zero sequence has probability $(2^{m-i}-1)/(2^m-1)$. Therefore, with a 23-stage MLFSR, an all-zero sequence of five digits will occur in the output with an approximate probability of 1/32. Since the all-zero sequence is easy to detect with a digital machine, and since it was assumed that the all-zero string of information bits was transmitted, the error pattern generator was programmed to insert an error bit (i.e., change a received symbol to a "1") each time the five-digit zero sequence occurred in the output of the MLFSR. This, then, happened with a nominal probability of 1/32.

2. Binary Symmetric Channel (p = 0.04688)

This channel with crossover probability 0.04688 = 3/64 was also simulated using a 23-stage MLFSR. In this case, each time the five-digit zero sequence occurred, the error pattern generator was programmed to insert an error bit (probability = 1/32). If the five-digit zero sequence did not occur, the generator then checked for a six-digit

sequence of zeros. If the six-digit sequence of zeros was present (probability = 1/64) the error bit was inserted. Therefore, the overall probability of occurrence of an error was [(1/32) + (1/64)] = (3/64).

LIST OF REFERENCES

1. Shannon, C. E., "A Mathematical Theory of Communication," Bell System Technical Journal, v. 27, p. 379-423, 623-656, 1948.

2. Forney, G. D., Jr., "Convolutional Codes I: Algebraic Structure," IEEE Transactions on Information Theory, v. IT-16, p. 720-738, 1970.

3. Costello, D. J., Jr., Construction of Convolutional Codes for Sequential Decoding, University of Notre Dame Technical Report No. EE-692, August 1969.

4. Wozencraft, J. M., "Sequential Decoding for Reliable Communication," IRE Convention Record, 1957, Part 2, p. 11-25.

5. Wozencraft, J. M. and Jacobs, I. M., Principles of Communication Engineering, Wiley, 1965.

6. Jacobs, I. M. and Berlekamp, E. R., "A Lower Bound to the Distribution of Computation for Sequential Decoding," IEEE Transactions on Information Theory, v. IT-13, p. 167-174, 1967.

7. Gallager, R. G., Information Theory and Reliable Communication, Wiley, 1968.

8. Massey, J. L., Threshold Decoding, The MIT Press, 1963.

9. Robinson, J. P., "Error Propagation and Definite Decoding of Convolutional Codes," IEEE Transactions on Information Theory, v. IT-14, p. 121-128, 1968.

10. Wozencraft, J. M. and Reiffen, B., Sequential Decoding, The MIT Press, 1961.

11. Gilbert, E. N., "A Comparison of Signalling Alphabets," Bell System Technical Journal, v. 31, p. 504-522, 1952.

12. Wagner, T. J., "A Gilbert Bound for Periodic Binary Convolutional Codes," IEEE Transactions on Information Theory, v. IT-14, p. 752-755, 1968.

13. Costello, D. J., Jr. and Morrissey, T. N., Jr., "Strengthened Lower Bound on Definite Decoding Minimum Distance for Periodic Convolutional Codes," IEEE Transactions on Information Theory, v. IT-17, p. 212-214, 1971.

14. Zigangirov, K. Sh., "Some Sequential Decoding Procedures," Problemy Peredachi Informatsii, v. 2, No. 4, p. 13-25, 1966.

15. Jelinek, F., "Fast Sequential Decoding Algorithm Using a Stack," IBM Journal of Research and Development, v. 13, p. 675-685, 1969.

16. Geist, J. M., Algorithmic Aspects of Sequential Decoding, University of Notre Dame Technical Report No. EE-702, August 1970.

17. Geist, J. M., "An Empirical Comparison of Two Sequential Decoding Algorithms," IEEE Transactions on Communication Technology, v. COM-19, p. 415-419, 1971.

18. Fano, R. M., "A Heuristic Discussion of Probabilistic Decoding," IEEE Transactions on Information Theory, v. IT-9, p. 64-74, 1963.

19. Forney, G. D., Jr., "Use of a Sequential Decoder to Analyze Convolutional Code Structure," IEEE Transactions on Information Theory, v. IT-16, p. 793-795, 1970.

20. Peterson, W. W., Error-Correcting Codes, The MIT Press, 1961.

21. Bahl, L. R. and Jelinek, F., "Rate ½ Convolutional Codes with Complementary Generators," IEEE Transactions on Information Theory, v. IT-17, p. 718-727, 1971.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Asst Professor J. M. Geist, Code 52 GJ Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
4.	LCDR Barry H. Lerich, USN Class 51 Armed Forces Staff College Norfolk, Virginia 23511	1
5.	Department of Electrical Engineering Code 52 EC Naval Postgraduate School Monterey, California 93940	1

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Naval Postgraduate School Monterey, California 93940	Unclassified
	2b. GROUP

3. REPORT TITLE

SEQUENTIAL DECODING OF TIME-VARYING CONVOLUTIONAL CODES

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Master's Thesis; December 1971

5. AUTHOR(S) *(First name, middle initial, last name)*

Barry Harold Lerich; Lieutenant Commander, United States Navy

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
December 1971	72	21

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

A discussion of convolutional encoding and sequential decoding is given as backgrojnd for the experimental phase of this research. A modified stack algorithm is programmed to simulate a sequential decoder for use in the study of time-varying convolutional codes.

The decoder simulator is used to analyze the structure of codes with the aim of finding a systematic method by which to construct time-varying convolutional codes with good decoding properties. Some difficulties in construction techniques are discussed.

Extensive computer simulation comparing decoding performance of time-varying codes to that of fixed codes is reported. The conclusion of these comparisons is that the time-varying codes studied thus far are of theoretical interest only.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

UNCLASSIFIED

Security Classification

A-31408

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
CONVOLUTIONAL CODES						
SEQUENTIAL DECODING						